Scheduling DAGs on Asynchronous Processors

Michael A. Bender[*]
Department of Computer Science
Stony Brook University
Stony Brook, NY 11794-4400, USA
bender@cs.sunysb.edu

Cynthia A. Phillips[†]
Discrete Algorithms and Math Department
Sandia National Laboratories
Albuquerque, NM, USA
caphill@sandia.gov

ABSTRACT

This paper addresses the problem of scheduling a DAG of unit-length tasks on ***asynchronous*** processors, that is, processors having different and changing speeds. The objective is to minimize the ***makespan***, that is, the time to execute the entire DAG. Asynchrony is modeled by an ***oblivious adversary***, which is assumed to determine the processor speeds at each point in time. The oblivious adversary may change processor speeds arbitrarily and arbitrarily often, but makes speed decisions independently of any random choices of the scheduling algorithm.

This paper gives bounds on the makespan of two randomized online ***firing-squad*** scheduling algorithms, All and Level. These two schedulers are shown to have good makespan even when asynchrony is arbitrarily extreme. Let W and D denote, respectively, the number of tasks and the longest path in the DAG, and let π_{ave} denote the average speed of the p processors during the execution.

In All each processor repeatedly chooses a random task to execute from among all ***ready*** tasks (tasks whose predecessors have been executed). Scheduler All is shown to have a makespan $T_p =$

- $\Theta\left(\frac{W}{p\pi_{ave}}\right)$, when $\frac{W}{D} \geq p \log p$
- $\Theta\left((\log p)^{\alpha}\frac{W}{p\pi_{ave}} + (\log p)^{1-\alpha}\frac{D}{\pi_{ave}}\right)$, when $\frac{W}{D} = p(\log p)^{1-2\alpha}$, for $\alpha \in [0,1]$
- $\Theta\left(\frac{D}{\pi_{ave}}\right)$, when $\frac{W}{D} \leq \frac{p}{\log p}$,

both expected and with high probability. A family of DAGs is exhibited for which this analysis is tight.

In Level each of the processors repeatedly chooses a random task to execute from among all ***critical*** tasks (ready tasks at the *lowest* level of the DAG). This second scheduler is shown to have a makespan of

$$T_p = \Theta\left(\frac{W}{p\pi_{ave}} + [\log^* p - \log^*(pD/W)]\,\frac{D}{\pi_{ave}}\right),$$

both expected and with high probability. Thus, Level is always at least as good as All asymptotically, and sometimes better.

Categories and Subject Descriptors

F.2.2 [**Analysis of Algorithms and Problem Complexity**]: Nonnumerical Algorithms and Problems – Sequencing and Scheduling

General Terms

Algorithms, Theory

Keywords

Asynchronous parallel computing, online scheduling, firing-squad scheduling, precedence-constrained scheduling

1. INTRODUCTION

We consider the problem of executing irregularly-structured (e.g., multithreaded) parallel programs on ***asynchronous*** processors, that is, processors running at different and changing speeds. Asynchrony can occur in any setting in which parallel resources are shared. For example, in grid computing, a remote user of a machine may have significantly reduced privileges compared to the machine owner, and the remote user's job may run sluggishly when the owner returns to a machine. In server farms, users with similar privileges must share the resources, so a machine's instantaneous speed for a user depends on current load.

We show how to achieve asymptomatically good performance bounds under arbitrarily extreme asynchrony. We assume that processors can run arbitrarily slowly, down to speed zero, and arbitrarily fast. Processor can change speeds arbitrarily frequently. Since the asynchrony can be adversarially extreme, dynamic forecasters of system-resource performance or useability such as NWS (network weather system) are unreliable. A scheduler in this setting cannot trust any current or historical system-performance estimates.

We model asynchrony by assuming an ***oblivious adversary***, which determines the processor speeds at every instant in time. The oblivious adversary knows the structure of the program, but is unaware of any random choices made by the task scheduler. This adversary models the case in which processor speeds are independent of the execution of

[*]Supported in part by NSF Grants CCF-0621439/0621425, CCF-0540897/05414009, CCF-0634793/0632838, and CNS-0627645.

[†]Sandia is a multipurpose laboratory operated by Sandia Corporation, a Lockheed-Martin Company, for the United States Department of Energy's National Nuclear Security Administration under contract DE-AC04-94AL85000.

the parallel program. This is a common case, since it models loads from other users, power outages, and other influences outside the control of the user. However, the adversary does not model all sources of asynchrony, only those that are independent of the random choices of the program.[1]

Firing-Squad Scheduling.

Our scheduler uses ***firing-squad scheduling*** to assign tasks to processors. In firing-squad scheduling, whenever a processor becomes free, it randomly and independently chooses some task to execute from a set of ***enabled tasks***. The enabled tasks are a subset of the ***ready tasks***, i.e., unfinished tasks that are ready to run. Thus, in firing-squad scheduling the only algorithmic choice is how to determine the set of enabled tasks. Firing-squad scheduling is a form of ***eager scheduling***, in which the same task may be executed simultaneously by many processors.

Firing-squad scheduling has advantages in dealing with extreme asynchrony. Because tasks can be executed redundantly, there is no need to preempt any process or migrate it to a faster processor. Once a task begins on a processor, it is executed to completion on that processor. Moreover, firing-squad schedulers can take advantage of "bursts" of computing speed, and firing-squad scheduling works well even when current and past speeds are uncorrelated.

In this paper, we consider firing-squad scheduling of unit-length tasks in an arbitrary directed acyclic graph (DAG). We study two natural ways to select enabled sets. First, we consider an enabled set consisting of *all* ready tasks. Second, we consider a restrictive choice of enabled set, which reduces the parallelism and locally increases the probability of redundant execution. We show that, counterintuitively, limiting parallelism this way can actually *reduce* the ***makespan***, that is, the completion time of the last task.

Much previous work in asynchronous parallel computing considers firing-squad and other eager-scheduling algorithms (see e.g., [2, 3, 4, 5, 7, 6, 8, 25, 26, 27, 28, 30, 31]). This prior work focuses on executing programs with full synchronization barriers, frequently PRAM programs. In such programs each task in the DAG in level $\ell + 1$ has an explicit precedence constraint with each task in level ℓ. To the best of our knowledge, we give the first asymptotic analysis of firing-squad scheduling in general DAGs.

Scheduling Model and Algorithms.

We now give definitions and terminology. A parallel program is modeled as a DAG $G = (V, E)$ of precedence-constrained tasks of unit size. The vertices represent the tasks, and the edges represent the dependencies between tasks. Let D denote the ***critical-path length***, that is, the length of the longest path in G. Let W denote the ***total work***, that is, the number of vertices, $|V|$, in G. Let $\pi_1(t), \pi_2(t), \ldots, \pi_p(t)$ denote the instantaneous speeds of the p processors at time t. Let $\pi_{ave}(t)$ denote the instantaneous average speed of all the processors at time t, i.e., $\pi_{ave}(t) = \sum_{i=1}^{p} \pi_i(t)/p$. For any given schedule having makespan T, let π_{ave} denote the average speed of the processors during the schedule, i.e., $\pi_{ave} = \int_{t=0}^{T} dt\, \pi_{ave}(t)/T$. The objective is to minimize the makespan. The ***level*** of a vertex is the longest path from the start of the DAG to that vertex. A task is ***ready*** if all its predecessors have been executed. A ready task is ***critical*** if its level is smallest from among all ready tasks. A ready task is ***enabled*** in a firing-squad scheduler if it is added to the task pool from which the processors randomly choose.

We now give the performance of greedy schedules on both homogeneous and heterogeneous processors. In a ***greedy schedule*** a global scheduler greedily assigns ready tasks to processors. The scheduler assures no redundancy in this assignment. The makespan T_p of a greedy schedule on p identical processors is $T_p \leq W/(p\pi_{ave}) + D/\pi_{ave}$ [21, 12]. Because both $W/(p\pi_{ave})$ and D/π_{ave} are lower bounds on the optimal makespan, T_p is a 2-approximation of the optimal makespan [21]. More recently, the makespan was analyzed when processors have different speeds. If at all times the scheduler runs the fastest processors (whenever k processors must be idle, these are the k slowest), then these same bounds apply [10, 9]. For heterogeneous processors, the schedule is ***preemptive*** and requires ***migration***. That is, if the amount of available work decreases or processors change speeds, then a task may need to be stopped on one processor and resumed on another, currently faster, processor. In general D/π_{ave} is no longer a lower bound for heterogeneous processors, and even for unvarying speeds, the best known approximation ratio is only $O(\log p)$ [16, 14].

Nonetheless, for the common case in parallel computing, we can give better approximation ratios. Typically, the average parallelism W/D is greater than p, that is, $W/p > D$. If not, then there are too many processors for the parallel program. When $W/p > D$, a makespan dominated by a $\Theta(W/p)$ term is nearly optimal, both for homogeneous and heterogeneous processors.

Analysis of Two Firing-Squad Schedulers.

We now formally describe our scheduling problem. The objective is to schedule a DAG $G = (V, E)$ of precedence-constrained tasks of unit size. The DAG is revealed online. There is no preemption. Processors are asynchronous and their speeds are determined by an oblivious adversary. Processors choose which task to execute using firing-squad scheduling. As described above, the combination of randomization and redundancy is enough to schedule without the need for task migration.

In much of the paper, we can begin our analysis assuming firing-squad scheduling on uniform-speed processors. We then show how these results carry over to asynchronous processors.

We analyze two variants of firing squad scheduling, All and Level. In All, the pool of enabled tasks is the entire set of *ready* tasks. In Level, the pool of enabled tasks is only the set of *critical* tasks, which can be much smaller.

At first glance, it is not clear which of these two algorithms is better or whether there is even a significant performance difference. We now make a case for each algorithm. For simplicity we temporarily assume that processors are synchronous. The advantage of All is that in any given step, the enabled-task pool is maximally large. Thus, the amount of redundant execution is minimized and the amount of completed work is maximized in any given time step. The advantage of Level is that the smaller enabled-task pool only contains tasks at the lowest level of the DAG, and as a result, progress can be made faster on the critical path. Prior algorithms for DAG scheduling (e.g., [19]) advocate a "work-

[1]For example, processes that crash machines affect processor speeds. An oblivious adversary could not model asynchrony caused by these machine crashes.

first principle." This principle says that it is better to amortize against the total work rather than against the critical path length. A reasonable interpretation of the work-first principle is that All has an advantage. In fact, as we show below, the makespan of Level is asymptotically better than the makespan of All.

Asymptotic Performance.

We give tight analysis of the firing-squad schedulers All and Level. We show that with high probability All has a makespan $T_p =$

- $\Theta\left(\frac{W}{p\pi_{ave}}\right)$, when $\frac{W}{D} \geq p \log p$
- $\Theta\left((\log p)^{\alpha}\frac{W}{p\pi_{ave}} + (\log p)^{1-\alpha}\frac{D}{\pi_{ave}}\right)$, when $\frac{W}{D} = p(\log p)^{1-2\alpha}$, for $\alpha \in [0,1]$
- $\Theta\left(\frac{D}{\pi_{ave}}\right)$, when $\frac{W}{D} \leq \frac{p}{\log p}$.

We also show that Level has a makespan of only

$$T_p = \Theta\left(\frac{W}{p\pi_{ave}} + [\log^* p - \log^*(pD/W)]\frac{D}{\pi_{ave}}\right).$$

The makespan is achieved with probability at least $1-\varepsilon$, where $\varepsilon = 1/\text{poly}(p)$ when $D = O(\text{polylog}(p))$ and $\varepsilon = 1/\exp(p)$ when $D = \Omega(\text{polylog}(p))$. We exhibit DAGs for which this analysis is tight.

We prove these results first for uniform-speed processors. Then we generalize for asynchronous processors. Thus, perhaps surprisingly, the makespan can be decreased by appropriately throttling the available parallelism.

2. PRELIMINARIES: FIRING-SQUAD SCHEDULING WITH SYNCHRONIZATION BARRIERS

This section analyzes firing-squad scheduling on DAGs with full synchronization barriers. In such DAGs all tasks at layer ℓ must be completed before any task at layer $\ell+1$ begins. Most previous bounds for firing-squad and eager scheduling apply only to these DAGs (see e.g., [27, 4, 3, 2, 5, 25, 30, 28, 26, 7, 6, 8, 31].

By including only critical tasks in the enabled pool, Level effectively transforms an arbitrary DAG into a synchronization-barrier DAG. As a result, much of the analysis for Level already appears in some form in the literature. In this paper we give a full analysis for completeness and to introduce techniques we use later. We believe that our analysis is cleaner, more general, and more complete than any that has appeared previously. However, our main point is to argue that by adding up to $O((W/D)^2D)$ additional task dependencies, we can actually reduce the makespan.

In a synchronous round of firing squad scheduling, the probability of task A being chosen is not independent of the probability of task B being chosen. For example, if we know that task A was chosen by processor P_A, then processor P_A did not choose task B, and therefore the probability of task B being executed is reduced. However, the task-selection probabilities are negatively correlated, defined as follows: Consider a set of t random binary variables $X_1, \ldots, X_t$. We say that the variables are ***negatively correlated*** if for all subsets $S \subseteq \{1 \ldots t\}$:

$$\Pr\left\{\bigwedge_{i\in S}(X_i = 0)\right\} \leq \prod_{i\in S}\Pr\{X_i = 0\}, \quad \text{and} \qquad (1)$$

$$\Pr\left\{\bigwedge_{i\in S}(X_i = 1)\right\} \leq \prod_{i\in S}\Pr\{X_i = 1\}. \qquad (2)$$

Our analysis requires an extension of Chernoff bounds for negatively correlated random variables due to Panconesi and Srinivasan[33, 34].

Theorem 1 ([33, 34]). *Let $a_1, a_2, \ldots, a_t$ be reals in $[0,1]$, and $X_1, X_2, \ldots, X_t$ be random binary variables where $X_i \in \{0,1\}$.*
(i) Suppose the random variables are negatively correlated in the 1 values (expression (2)) and $E[\sum_i a_iX_i] \leq \mu_1$. Then for any $\delta \geq 0$

$$\Pr\left\{\sum_i a_iX_i \geq \mu_1(1+\delta)\right\} \leq \left(\frac{e^{\delta}}{(1+\delta)^{1+\delta}}\right)^{\mu_1}. \qquad (3)$$

(ii) Suppose the random variables are negatively correlated in the 0 values (expression (1)) and $E[\sum_i a_iX_i] \geq \mu_2$. Then for any $\delta \in \{0,1\}$

$$\Pr\left\{\sum_i a_iX_i \leq \mu_2(1-\delta)\right\} \leq e^{-\mu_2\delta^2/2}. \qquad (4)$$

We now show how long it takes p synchronous processors (with uniform speed) to execute m *independent* unit-length tasks:

Theorem 2. *Consider n unit-length enabled tasks executed on p processors using firing-squad scheduling. If the processors run synchronously at uniform speed $\pi_1(t), \pi_2(t), \ldots, \pi_p(t) = \pi_{ave}$, then the makespan is $\Theta\left(n/(p\pi_{ave}) + [\log^* p - \log^*(p/n)]/\pi_{ave}\right)$ with probability at least $1 - \min\left\{2^{-\Theta(\sqrt{p})}, 2^{-\Theta(n/p)}\right\}$.*

Proof. We divide the execution into two phases. Phase 1 begins at time $t=0$ and ends once the number of unexecuted tasks dips below p. Phase 2 begins at the end of Phase 1 and ends once all tasks have completed.

We first show that Phase 1 contains $\Theta(n/p)$ steps with probability at least $1 - \min\left\{2^{-\Theta(p)}, 2^{-\Theta(n/p)}\right\}$. Consider a time step t when there are $m \geq p$ remaining jobs. For time step t, define 0/1-random variable

$$X_i = \begin{cases} 1 & \text{if job } i \text{ is executed;} \\ 0 & \text{otherwise.}\end{cases}$$

Then

$$\begin{aligned}\Pr\{X_i = 0\} &= (1-1/m)^p \qquad (5)\\ &= (1-1/m)^{m\cdot p/m}\\ &\leq e^{-p/m}.\end{aligned}$$

For any such time step t, define random variable $X = \sum_{i=1}^{m} X_i$ to be the total number of jobs executed during t. Then,

$$\mathrm{E}[X] \geq m\left(1 - e^{-p/m}\right).$$

Because $e^{-x} \leq 1 - x + x^2/2$ when $0 \leq x \leq 1$,

$$\begin{aligned} \mathrm{E}[X] &\geq m\left(p/m - p^2/2m^2\right) \\ &\geq p/2\,. \end{aligned}$$

We bound X from below, showing that in phase 1, $X = \Theta(p)$ with very high probability. The X_i random variables are not independent. However, they are negatively correlated (see (1) and (2)):

LEMMA 3. *The X_i random variables, indicating selection during a synchronous round of firing squad scheduling, are negatively correlated.*

PROOF. We first prove by induction on $|S|$ that (1) and (2) hold. The base case that $|S| = 1$ holds trivially. Next we prove the inductive step. Assume for the sake of induction that (1) and (2) hold for all sets S such that $|S| = k$. Now let $S = \{j\} \cup S'$, where $|S'| = k$ and $|S| = k+1$ (i.e., $j \notin S'$).

First we calculate the probability that $X_j = 0$, given that at least k other jobs are not executed:

$$\begin{aligned} \Pr\left\{X_j = 0 | \bigwedge_{i \in S'} (X_i = 0)\right\} &= [1 - 1/(m-k)]^p \\ &\leq (1 - 1/m)^p \\ &= \Pr\{X_j = 0\}\,. \end{aligned}$$

Therefore, we prove the inductive step establishing (1) as follows:

$$\begin{aligned} &\Pr\left\{\bigwedge_{i \in S} (X_i = 0)\right\} \\ &\quad = \Pr\left\{X_j = 0 | \bigwedge_{i \in S'} (X_i = 0)\right\} \Pr\left\{\bigwedge_{i \in S'} (X_i = 0)\right\} \\ &\quad \leq \prod_{i \in S} \Pr\{X_i = 0\}\,. \end{aligned}$$

We perform a similar induction establishing (2) since

$$\begin{aligned} \Pr\left\{X_j = 1 | \bigwedge_{i \in S'} (X_i = 1)\right\} &= 1 - (1 - 1/m)^{p-k} \\ &\leq 1 - (1 - 1/m)^p \\ &= \Pr\{X_j = 1\}\,. \end{aligned}$$

□

We now argue that firing-squad scheduling completes $X = \Theta(p)$ jobs in a single time step with high probability when there are $m \geq p$ enabled jobs in the system. There are at most p jobs executed because there are only p processors. Thus $X = O(p)$. By Theorem 1, Inequality (4), $\Omega(p)$ jobs are executed with probability at least $1 - 2^{-\Theta(p)}$. To increase the magnitude of the exponent, consider α rounds. In α rounds, the probability that at least $\Theta(p)$ jobs are executed is at least $1 - 2^{-\Theta(\alpha p)}$. We explicitly represent parameter α even when it is a constant. Thus, we illustrate the tradeoff between increased makespan and decreased error probability: a multiplicative increase in makespan yields an exponential decrease in failure probability.

We now show that Phase 1 completes in $O(1 + n/p)$ steps with high probability. It is sufficient to have $\Theta(n/p)$ steps during which $\Theta(p)$ tasks are executed. We say that a group of c rounds (a ***superstep***) is ***good*** if at least $\Theta(p)$ jobs are executed during that time step, and ***bad*** otherwise. The probability that a time step is good is at least 1/2 for large enough constant c (since it is at least $1 - 2^{-\Theta(cp)}$). By ordinary Chernoff bounds, at least a constant fraction of $\alpha n/p$ supersteps are good with probability at least $1 - 2^{-\Theta(\alpha n/p)}$.

Alternatively, by the union bound, the probability that any of n/p supersteps, each of length α, fails to execute $\Theta(p)$ tasks is at most $(n/p)2^{-\Theta(\alpha p)}$. Thus, with probability $1 - n2^{-\Theta(\alpha p)}/p = 1 - 2^{(\lg(n/p) - \Theta(\alpha p))}$, Phase 1 completes in $O(\alpha n/p)$ rounds. If $\log(n/p) = O(p)$, then we set α to a constant large enough to dominate the positive term in the exponent. When $\log(n/p) = \Omega(p)$, the direct union bound is not useful. But in this case n/p so dominates p, that $2^{-\Theta(n/p)}$ is asymptotically smaller than $2^{-\Theta(p)}$. Therefore, combining these two arguments, for suitably-large α, the probability that Phase 1 completes in time $\Theta(1 + \alpha n/p)$ is at least $1 - \min(2^{-\Theta(\alpha p)}, 2^{-\alpha n/p})$.

We now bound the length of Phase 2. Suppose that Phase 2 begins with n' ($n' < p$) remaining tasks. We show that Phase 2 has length $O(\log^* p - \log^*(p/n'))$ with probability at least $1 - 2^{-\Theta(\sqrt{p})}$. More generally, we show that for constant α, Phase 2 has length $O(\alpha\,[\log^* p - \log^*(p/n')])$ with probability at least $1 - 2^{-\Theta(\alpha\sqrt{p})}$.

Define $\mathrm{tower}(x,i) = \underbrace{x^{x^{\cdot^{\cdot^{x}}}}}_{i}$ for integer $i \geq 1$ and define $\mathrm{tower}(x,0) = 1$.

We prove that in time step t, as long as the number of remaining jobs $m_t \leq p/\mathrm{tower}(2,k)$ in the system is sufficiently large (specified below) then the following holds: With probability at least $1 - 2^{-\Theta(\sqrt{p})}$, in the next time step $t+1$, the number of remaining jobs is $m_{t+1} \leq p/\mathrm{tower}(2,k+1)$. Moreover, with probability at least $1 - 2^{-\Theta(\alpha\sqrt{p})}$, after α rounds in a time step, the number of remaining jobs is at most $p/\mathrm{tower}(2,k+1)$.

In time step t of Phase 2, where there are m jobs remaining, we let 0/1-random variables $R_i^{(t)}$ be defined as follows:

$$R_i^{(t)} = \begin{cases} 1 & \text{if job } i \text{ remains after step } t; \\ 0 & \text{otherwise.} \end{cases}$$

Let random variable $R^{(t)} = m_{t+1} = \sum_{i=1}^{m} R_i^{(t)}$ denote the number of remaining jobs at the end of step t. The $R_i^{(t)}$ random variables are negatively correlated since the X_i random variables above are negatively correlated. We have:

$$\begin{aligned} \Pr\left\{R_i^{(t)} = 1\right\} &= (1 - 1/m_t)^p \\ &\leq [1 - \mathrm{tower}(2,k)/p]^p \\ &\leq e^{-\mathrm{tower}(2,k)}\,. \end{aligned}$$

Thus, the expected number of remaining jobs is

$$\mathrm{E}\left[R^{(t)}\right] \leq m_t e^{-\mathrm{tower}(2,k)},$$

which is bounded away from $p/\mathrm{tower}(2,k+1)$ by at least a constant factor.

Thus, as long as $m_t e^{-\mathrm{tower}(2,k+1)} \geq \sqrt{p}$, we can use Chernoff bounds (Theorem 1) to show that with probability at least $1 - 2^{-\Theta(\sqrt{p})}$, the number of jobs in the next round, m_{t+1} (which equals $R^{(t)}$), is at most $p/\mathrm{tower}(2,k+1)$.

If not, then the "endgame" begins. By Chernoff bounds, with probability at least $1 - 2^{-\Theta(\sqrt{p})}$, the number of jobs in the next round $m_{t+1} \leq \sqrt{p}$. We next show that if $m_{t+1} \leq \sqrt{p}$, then $m_{t+2} < 1$ (and therefore 0) with probability at least $1-2^{-\Theta(\sqrt{p})}$. This last step follows by the union bound, since $\Pr\{R_j = 1\}$, the probability that any given job j survives, is at most $\left(1 - 1/\sqrt{p}\right)^p \leq e^{-\sqrt{p}}$.

We now show how this intermediate result leads to the claimed theorem. Let function TowerRnd(m) round m up to the nearest value of $p/\text{tower}(2,k)$, for integer $k \geq 0$. Let steps(m) be that particular value of k. If $n' = p$, then with probability at least $1 - 2^{-\Theta(\alpha\sqrt{p})}$, Phase 2 takes $O(\alpha \log^* p)$ rounds because by the definition of $\log^* p$, we have $p/\text{tower}(2, \log^* p) \leq 1$. If $\text{TowerRnd}(n') < p$, i.e., $\text{steps}(n') > 0$, then Phase 2 takes only $O(\alpha\,[\log^* p - \text{steps}(n')])$. If

$$p/\text{tower}(2, k+1) \leq n' \leq p/\text{tower}(2,k),$$

then

$$\text{tower}(2,k) \leq p/n' \leq \text{tower}(2, k+1),$$

which means that

$$\log^*(p/n') = \Theta(k) = \Theta(\text{steps}(n'))$$

as promised.

There is one final issue in computing the overall error probability. The Phase 2 error probability may be much larger than that of Phase 1. In this case, choose $\alpha = \Theta(n/p^{3/2})$ for Phase 2. This increases the makespan of phase 2 to $O(\alpha \lg^* p)$, but this is still smaller than the Phase 1 makespan of $\Theta(n/p)$. Thus the total makespan increases by at most a constant factor.

We now show how Theorem 2 adapts when processors have different speeds.

THEOREM 4. *Consider n unit-length enabled tasks executed on p asynchronous processors using firing-squad scheduling. The makespan is $\Theta\left(n/(p\,\pi_{ave}) + [\log^* p - \log^*(p/n)]/\pi_{ave}\right)$, where π_{ave} is the average speed of the processors during the execution.*

PROOF. We partition the execution into ***stages***, defined as follows. Stage 0 begins at time 0. Each stage k ends once there have been at least $3p$ units of work completed entirely within the stage. At least p tasks are entirely executed during the stage because at most p tasks can overlap between stage $k-1$ and stage k and at most p tasks can overlap between stage k and stage $k+1$. For the analysis, we ignore all work done on overlapping tasks; thus, we only take advantage of a third of the processors' random choices. We use an analysis similar to that of Theorem 2. For each random execution, the probability that it is redundant is less than or equal to the probability that it is redundant in the synchronous case. □

THEOREM 5. *Consider a DAG G with critical path D and total work W executed on p asynchronous processors using* LEVEL *firing-squad scheduling. Then the makespan is*

$$T_p = \Theta\left(\frac{W}{p\pi_{ave}} + [\log^* p - \log^*(pD/W)]\,\frac{D}{\pi_{ave}}\right)$$

where π_{ave} is the average speed of the processors during the execution.

PROOF. Let $W = n_1 + n_2 + \cdots + n_D$, where n_ℓ is the number of tasks on the ℓth level of the DAG. We show by an exchange argument that the makespan is maximized when all n_ℓ are all within an additive one of each other.

We now define terms. For simplicity, we discuss the synchronous case. As in the proof of Theorem 4, the synchronous case is easily transformed into the asynchronous case.

The execution is divided into phases. During phase ℓ, the n_ℓ tasks at level ℓ of the DAG are executed. Let length(ℓ) be the length of phase ℓ. Let e_t be the maximum number of independent tasks that can be executed in t time steps w.h.p. (see Theorems 2 and 4).

Now suppose for the sake of contradiction that there is no way to maximize the makespan while keeping all phase lengths within one of each other. Consider an optimal solution such that the maximum difference δ between phase lengths is minimized, i.e., a solution that minimizes $\delta = \max_{i,j} |\text{length}(i) - \text{length}(j)|$. Among all such optimal solutions, consider an optimal solution that minimizes the number of pairs (i,j) such that $\delta = |\text{length}(i) - \text{length}(j)|$.

Now we consider one such pair of phases (i,j) such that $\delta = \text{length}(i) - \text{length}(j)$. We show how to reduce n_i and increase n_j so that the makespan does not decrease. Reduce n_i by the maximum amount x such that length(i) decreases by one. If we increase n_j by x, then length(j) increases by at least one. This follows because $e_{t+1} - e_t$ is monotonically increasing in t. Thus, we have found a new optimal solution in which the number of maximal pairs is decreased by one, contradicting our assumption. □

3. UPPER BOUND FOR THE ALL SCHEDULING ALGORITHM

We now analyze the ALL scheduling algorithm. Recall that in ALL, each processor randomly chooses one task to execute from among all enabled tasks. In any given time step, ALL has less redundancy than LEVEL, but it is globally suboptimal. This section gives an upper bound on the makespan of ALL. The next section gives a matching lower bound.

We establish the following upper bound:

THEOREM 6. *Algorithm* ALL *runs in time*

- $\Theta\left(\dfrac{W}{p\pi_{ave}}\right)$, *when* $\dfrac{W}{D} \geq p\log p$
- $\Theta\left((\log p)^{\alpha}\dfrac{W}{p\pi_{ave}} + (\log p)^{1-\alpha}\dfrac{D}{\pi_{ave}}\right)$, *when* $\dfrac{W}{D} = p(\log p)^{1-2\alpha}$, *for* $\alpha \in [0,1]$
- $\Theta\left(\dfrac{D}{\pi_{ave}}\right)$, *when* $\dfrac{W}{D} \leq \dfrac{p}{\log p}$.

This bound holds with probability $1 - Dp^{-O(1)}$ if $D \leq \log^k p$ for some k and with probability $1 - 2^{-\Theta(D)}$ otherwise.

PROOF. We consider two types of time steps. In ***dense time steps*** the number of enabled tasks is greater than p. In ***sparse time steps*** the number of enabled tasks is at most p.

We bound the number of dense time steps in the following:

LEMMA 7. *The number of dense time steps is at most $O(\alpha W/p)$ with probability at least $1 - 2^{-\Theta(\alpha p + \alpha W/p)}$.*

PROOF. We first show that in a given dense time step, $\Theta(p)$ tasks are completed with probability at least $1-2^{-\Theta(p)}$, and more generally, in α rounds with at least p enabled tasks in each step, at least $\Theta(p)$ tasks are completed with probability at least $1 - 2^{-\Theta(\alpha p)}$. This claim follows from similar arguments to those in Theorem 2. In each time step, we have 0/1 random variables that are not independent, but are negatively correlated. Thus, we can still use Chernoff bounds (see Theorem 1).

As in Theorem 2, we define dense steps as good and bad, where a step is good if $\Theta(p)$ tasks are completed. Each step is good with at least a constant probability. Therefore, by ordinary Chernoff bounds, in $\Theta(\alpha W/p)$ time steps, there are at least W/p good time steps, with probability at least $2^{-\Theta(\alpha W/p)}$. After $\Theta(W/p)$ good steps, all work is complete. $\square$

We now consider only the ***sparse execution***, that is, the execution with all dense time steps removed. We partition the sparse execution into ***phases***, defined so that in phase j all the critical jobs in the DAG have depth j. Observe that the number of critical tasks in a phase is monotonically decreasing, whereas the number of enabled tasks can increase or decrease.

We first give this claim, which will be useful below:

LEMMA 8. *Let $x_1, x_2, \ldots, x_t \geq 0$ be constrained so that $\sum_{i=0}^{t} x_i = K$ for $K \geq 0$. Function $f(x_1, \ldots, x_t) = e^{-p/x_0}e^{-p/x_1} \ldots e^{-p/x_t}$ is maximized when $x_0 = x_1 = \cdots = x_t = K/t$.*

PROOF. We maximize function $f(x_1, \ldots, x_t) = e^{-p/x_0}e^{-p/x_1} \ldots e^{-p/x_t}$ by minimizing function $g(x_1, \ldots, x_t) = 1/x_0 + 1/x_1 + \cdots + 1/x_t$.

We will show that for any two variables x_i and x_j and any values of their positive sum $K_{ij} = x_i + x_j$, the sum $1/x_i + 1/x_j$ is minimized when $x_i = x_j$. Thus, we can minimize $1/x_i + 1/x_j = 1/x_i + 1/(K_{ij} - x_i)$ for any positive value of K_{ij}. Differentiating and setting to 0 we obtain $-1/x_i^2 + 1/(K_{ij} - x_i)^2 = 0$. The solution where both variables are positive is $x_i = x_j = K_{ij}/2$, as promised. Taking a second derivative we obtain $2/x_i^3 + 2/(K_{ij} - x_i)^3$, which is positive when $x_i = x_j = K_{ij}/2$, showing that we have a minimum.

Since for all pairs of values and possible sums, the sum of multiplicative inverses is minimized when the values are are equal, the lemma follows. $\square$

We now bound the probability that a given critical task survives for t time steps. We give this probability as a function of the number of enabled tasks during the t steps.

LEMMA 9. *The probability that a critical job survives for t (sparse) steps after the beginning of phase ℓ in the sparse execution, given that there are r_i enabled tasks in the ith step, for $i = 1 \ldots t$, is at most $e^{-p/r_1}e^{-p/r_2} \cdots e^{-p/r_t}$.*

PROOF. The probability that a given critical job survives for t (sparse) steps after the beginning of a phase is the probability that it survives the first round, times the probability that it survives the second, times the probability that it survives the third, etc., which is $(1 - 1/r_1)^p\,(1 - 1/r_2)^p \ldots (1 - 1/r_t)^p \leq e^{-p/r_1}e^{-p/r_2} \ldots e^{-p/r_t}$. $\square$

COROLLARY 10. *The probability that phase ℓ of the sparse execution lasts more than t steps, given that there are r_i enabled tasks in the ith step, for $i = 1 \ldots t$, is at most $pe^{-t^2 p/\left(\sum_{i=1}^{t} r_i\right)}$.*

PROOF. By Lemma 9, the probability that a given critical task survives for t time steps, given that the number of enabled tasks in step 1 through t is r_1 through r_t, respectively, is at most $e^{-p/r_1}e^{-p/r_2} \cdots e^{-p/r_t}$. By Lemma 8, this probability is maximized when all r_i are equal, meaning that the probability is at most $e^{-pt^2/\left(\sum_{i=1}^{t} r_i\right)}$. By the definition of a sparse step, there are at most p critical tasks at the beginning of any round. By the union bound, the probability that any of the at most p critical task survives is at most $pe^{-pt^2/\left(\sum_{i=1}^{t} r_i\right)}$, which is an upper bound on the probability that the round continues after t steps. $\square$

COROLLARY 11. *Suppose that for t time steps starting from the beginning of phase ℓ, the average number of enabled tasks per time step is r. If $t \geq r(c+1)\ln p/p$, then all critical tasks are completed (meaning phase ℓ has finished) with probability at least $1 - p^{-c}$.*

PROOF. This corollary follows immediately from Corollary 10 by plugging in values:

$$\begin{aligned}
\Pr\{\text{Still in phase } \ell\} &\leq pe^{-pt^2/\left(\sum_{i=1}^{t} r_i\right)} \\
&= pe^{-pt/r} \\
&\leq pe^{-p[r(c+1)\ln p/p]/r} \\
&= pe^{-(c+1)\ln p} \\
&= p^{-c}.
\end{aligned}$$

$\square$

The following lemma relates the total number of enabled tasks in a round to the total number of enabled tasks in each time step, summed over all time steps. (That is, suppose that a task gets a red ticket when it is first enabled and gets a blue ticket for each step that it is enabled. Then the lemma relates the number of red tickets to the number of blue tickets.)

LEMMA 12. *Suppose that in a phase of length t, there are a total of K tasks enabled during at least one step of the phase. Then with probability at least $1 - e^{-\Theta(\sqrt{p})}$,*

$$\sum_{i=1}^{t} r_i = \Theta(K).$$

PROOF. The probability that a given enabled task is executed in any sparse time step is at least $1 - (1 - 1/p)^p \geq 1 - 1/e$ because, by definition, a sparse steps contains at most p enabled tasks.

Suppose that there are fewer than $\sqrt{p}$ enabled tasks. Then, the probability that a given task is not executed is at most

$$(1 - 1/\sqrt{p})^p \leq e^{-\sqrt{p}}\,.$$

Therefore, by the union bound, the probability that even one task is not executed is at most $\sqrt{p}e^{-\sqrt{p}}$, which is exponentially small, as promised.

Now suppose that there are $p \geq r \geq \sqrt{p}$ enabled tasks in a given time step. We now give the setup for Chernoff bounds, similar to that in Theorem 2, showing that at least a constant fraction of enabled tasks are successfully executed, with very high probability. The probability that a given task is executed is at least $1 - 1/e$. Therefore, the expected number of executed tasks is at least $r(1-1/e)$. Let X_i be the 0/1-random variable which is 1 if task i is executed during the time step and 0 otherwise. Now define $X = \sum_{i=1}^{r} X_i$ to be the total number of executed jobs during the time step. Then $\mathrm{E}[X] \geq r(1 - 1/e)$.

The X_i variables are not independent, but they are negatively correlated, which is a sufficient condition for using Chernoff bounds (see Theorem 1). Plugging in Chernoff bounds with $\delta = 1/2$, the probability that there are fewer than $\mathrm{E}[X]/2$ critical jobs executed is at most $e^{-\sqrt{p}/32}$, which is also exponentially small in $\sqrt{p}$. □

Now given a fixed budget K of enabled jobs for a phase, we want to maximize the time spent in a single phase. We show that regardless of how the jobs are distributed per time step, after $O(\sqrt{K \log p/p})$ steps, all critical jobs are completed, meaning that the phase has ended.

Lemma 13. *Suppose that a phase contains at most K enabled jobs. Then the phase contains at most $O\left(\left\lceil \sqrt{K \log p/p} \right\rceil\right)$ time steps with probability $1 - p^{-O(1)}$.*

Proof. We exhibit a value of t such that the probability that the phase is still alive after t time steps is small. Let K be the total number of enabled tasks in all time steps and let K' be the total number of enabled jobs per time step, summed over all time steps. (Thus, K' counts an enabled job once for each time step that it is in the system, and K counts the number of "tickets" as described above.) By Lemma 12, $K = \Theta(K')$ with probability at least $1 - e^{-\Theta(\sqrt{p})}$.

By Corollary 11, with probability at least $1 - p^{-c}$, after $t \geq (K'/t)(c+1)\ln p/p$ time steps, all critical jobs are finished. Solving for t, it suffices to have $t^2 = \Theta(K(c+1)\ln p/p)$ or to have $t = \Theta\left(\left\lceil \sqrt{(c+1)K \ln p/p} \right\rceil\right)$. Therefore, after $t = \Theta\left(\left\lceil \sqrt{(c+1)K \ln p/p} \right\rceil\right)$ steps with a budget of K enabled jobs, the phase has completed. □

We now consider multiple phases. For the next few lemmas, we assume that the critical path D is polynomial in p, and then we remove this assumption. Define the ***sparse makespan*** to be the total number of sparse time steps in the execution. Suppose that phase i contains K_i enabled tasks, for $i = 1 \ldots D$. Then by using Lemma 13 and summing over all phases, we see that the sparse makespan is at most

$$O\left(\sum_{i=1}^{D} \left(1 + \sqrt{K_i \log p/p}\right)\right) \qquad (6)$$

with probability $1 - Dp^{-O(1)}$. We wish to maximize (6) over all choices of K_i. To do so, we use the following lemma, which follows from the Cauchy-Schwartz inequality:

Lemma 14. *The function $\sum_{i=1}^{n} \sqrt{x_i}$ subject to $\sum_{i=1}^{n} x_i = X$ is maximized when $x_1 = x_2 = \cdots = x_n = X/n$.*

Proof. The Cauchy-Schwartz inequality says that

$$|\mathbf{u} \cdot \mathbf{v}| \leq \|\mathbf{u}\| \cdot \|\mathbf{v}\|,$$

or equivalently,

$$\sum_{i=1}^{n} u_i v_i \leq \sqrt{\sum_{i=1}^{n} u_i^2} \sqrt{\sum_{i=1}^{n} v_i^2}\,.$$

It suffices to let $v_i = 1$ and $u_i = \sqrt{x_i}$ and the claim follows. □

The next lemma follows directly from (6) and Lemma 14.

Lemma 15. *Let $K = \min\{pD, W\}$. Then the sparse makespan is at most $O\left(\sqrt{KD \log p/p}\right)$ with probability $1 - Dp^{-O(1)}$.*

Now we can use Lemmas 7 and 15 to bound the makespan:

Lemma 16. *Let $K = \min\{pD, W\}$. Then the makespan is at most $O\left(W/p + \sqrt{KD \log p/p}\right)$ with probability $1 - Dp^{-O(1)}$.*

Now we show how to achieve better bounds as long as D is at least polylogarithmic in p. As before we divide into dense steps and sparse steps. By Chernoff bounds, the number of dense steps is bounded as in Lemma 7. We now divide the sparse steps into phases of length polylogarithmic in p. Each phase has length so that the critical path should decrease by at least one with probability at least $1/\mathrm{poly}(p)$ (see Lemma 13). A phase is good if the critical path does decrease, and it is bad otherwise. Now we can apply Chernoff bounds, showing that at least a constant fraction of phases are good. The probability of error is at most $2^{-\Theta(D)}$, and therefore we get the claimed makespan with probability at least $1 - 2^{-\Theta(D)}$. This is superior to the earlier bounds as long as D is at least polylogarithmic in p.

4. LOWER BOUND FOR THE ALL SCHEDULING ALGORITHM

In this section we give a matching lower bound for the makespan of All. Our approach is to exhibit a family of "shark-tooth graphs," which match the performance described in Section 3.

Figure 1 illustrates a ***shark-tooth graph***. Successors are *above* their ancestors in this example. A shark-tooth graph is built from several components. First is a series-parallel DAG. which we call the ***jaws*** of the shark-tooth graph. The ***first jaw*** includes a node u_1, which is the root of the DAG. For parameter y to be set later, this node u_1 has y immediate successors $u_{2,1}, u_{2,2}, \ldots, u_{2,y}$ called *spindle* elements or spindle nodes, also part of the first jaw. (The lower-bound construction will be most interesting when $\Omega(p/\log p) \leq y \leq O(p)$.) Each of these y successors is a direct predecessor of a single node u_3, which is part of the ***second jaw***. Node u_3 has y immediate successor spindle elements $u_{4,1}, u_{4,2}, \ldots, u_{4,y}$, also part of the second jaw. These nodes are predecessors to the third jaw, fourth jaw, etc, until node u_D, the deepest node in the DAG.

Each jaw also has "shark teeth." In shark dentistry, when one shark tooth falls out, there are many backup teeth behind, which move forward to fill the gap. In the shark-tooth

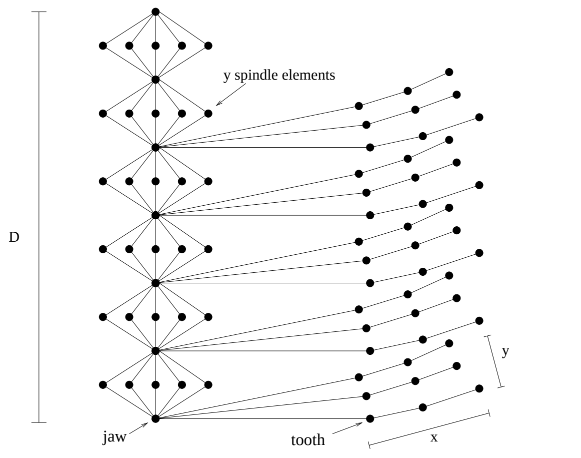

Figure 1: A shark-tooth DAG. Precedence relations are bottom up, so successors are above (or to the right of) their predecessors.

graph, each shark jaw holds y teeth at a time. Behind each tooth are x backup teeth, where x is another parameter to set later. More specifically, there are ***shark-tooth paths*** of length x of shark teeth, extending out of nodes u_{2i+1} for all $i < (D-x)/2$. (We need this restriction so that D truly is the critical-path length.)

We now describe how this graph G would be executed. Assume for simplicity that all processors run at the same speed π so that a processor executes one job per time step. Consider the time step t during which node u_{2i-1}, the beginning of the ith jaw, is executed. In the next time step, $t+1$, all y of the immediate (spindle) successors in the jaw, $u_{2i,1} \ldots u_{2i,y}$, are enabled and the first tooth of each of the y shark-tooth paths of u_{2i-1} is also enabled. The rest of the spindle nodes in the jaw $u_{2i,1} \ldots u_{2i,y}$ are predecessors of jaw $i+1$. In contrast, the shark-teeth paths are predecessors of no other parts of the graph, and it is therefore less urgent for the shark-teeth paths to be executed rapidly.

For fast firing-squad execution, therefore, the spindle nodes $u_{2i,j}$ of the jaw should be completed with higher priority, as analyzed in Section 2, yielding $\log^* p$ multiplicative overheads or better. However, the sharks teeth nodes are distracting to the processors, reducing the probability that the critical spindle jaw nodes are executed in a given round, and as we show, asymptotically increasing the makespan.

In the following we give a lower bound s on the number of steps needed for all of $u_{2i,1} \ldots u_{2i,y}$ (the rest of jaw i) to be executed (both expected and w.h.p.), under the assumption that none of the y shark-teeth paths are fully executed. In order to guarantee the condition that none of the shark-teeth paths are fully executed, we set one of the parameters, requiring that $x \geq s$.

Condition 1. *Let s be the target lower bound on the number of steps needed to execute each jaw. We set $x \geq s$. This ensures that none of the y shark-teeth paths will complete before s rounds.*

We can now bound the number of jaw nodes that are completed in a single round.

Lemma 17. *Suppose that at time step t there are m unexecuted vertices in jaw i. Then at time step $t+1$ there are at least $5^{-p/y}m$ unexecuted vertices in jaw i, with probability at least $1 - 2^{-\Theta(4^{-p/y}m)}$.*

Proof. Because there are y shark-tooth paths, regardless of how many jaw nodes remain, the probability that in a single time step, a given task $u_{2i,j}$ is *not* chosen is at least

$$\Pr\{u_{2i,j} \text{ not executed}\} \quad \geq \quad (1-1/y)^p\,.$$

Now, as in earlier proofs, we describe the problem in terms of 0/1-random variables. Consider the beginning of time step t, when there are m unexecuted vertices in the jaw. We define 0/1 random variables R_i as follows:

$$R_i = \begin{cases} 1 & \text{if job } i \text{ remains after step } t; \\ 0 & \text{otherwise.} \end{cases}$$

We let random variable $R = \sum_{i=1}^{m} R_i$ denote the total number of remaining jobs at the end of step t. If there are m unexecuted vertices in the jaw in time step t, then the expected number of remaining vertices in time step $t+1$, $\mathrm{E}[R]$, is at least

$$\begin{aligned} \mathrm{E}[R] &= m(1-1/y)^p \\ &= m(1-1/y)^{y \cdot p/y} \\ &> 4^{-p/y}m\,. \end{aligned}$$

We now use Chernoff bounds. The R_i random variables are not independent, but they are negatively correlated (see the proof of Theorem 2), thus permitting an application of Theorem 1. Then the probability that $R \leq 5^{-p/y}m$ is less than $2^{-\Theta(4^{-p/y}m)}$. (Here 5 could be replaced by any constant greater than 4.) $\square$

Lemma 18. *Consider the time step t during which node u_{2i-1}, the beginning of the ith jaw, is executed. With probability at least $1 - 2^{-\Theta(\sqrt{p})}$, at least $\Omega(1 + y \log y/p)$ steps are required to execute the tasks in jaw i.*

Proof. By Lemma 17, we are reducing the number of jaw nodes by a factor of at most $5^{p/y}$ in each round. In order to get better error bounds, we consider how long it takes until there are fewer than $O(\sqrt{p})$ jobs remaining, thus obtaining error probabilities that are at most $2^{-\Theta(\sqrt{p})}$.

The number of rounds is at least

$$\begin{aligned} s &= \Omega(\log_{5^{p/y}} y) \\ &= \Omega(\log y / \log(5^{p/y})) \\ &= \Omega(y \log y/p). \end{aligned} \tag{7}$$

We can also claim that $s \geq 1$ since each jaw is a 2-level DAG. Thus, we obtain the desired bounds. $\square$

We now explain why the lower bound is most interesting when $y = \Omega(p/\log p)$. When $y = cp/\log p$, for sufficiently small constant c, then all of the tasks $u_{2i,1} \ldots u_{2i,y}$, the remainder of jaw i, are executed in a single round with probability at least $1 - 1/\mathrm{poly}(p)$; each task is in fact executed simultaneously by $\Theta(\log p)$ processors. Consequently, $x = \Theta(1)$.

We are now ready to set the parameters x, the length of the shark-tooth path, and y, the number of shark-tooth paths per jaw. Our objective is to maximize s subject to Condition 1. We call such shark-tooth graphs ***maximal***.

For the sake of intuition, We begin with a special case, as described in the following lemma:

Lemma 19. *Consider a shark-tooth graph that is constrained such that $xy = \Theta(p)$. Then $s = x$, a lower bound on* *the number of steps necessary to execute a jaw, is maximized when $s = x = \Theta(\sqrt{\log p})$ and $y = \Theta(p/\sqrt{\log p})$.*

Proof. From Lemma 18, there exists some constant c such that

$$cx \geq y \log p/p.$$

Moreover, x and y are constrained such that

$$xy = p.$$

Substituting for x we obtain

$$cp^2/\log p \geq y^2,$$

and the constraint on y follows by taking square roots. $\square$

Now we give the full tradeoff of x and y:

Lemma 20. *Consider a shark-tooth graph that is constrained such that $xy = \Theta(p \log^{2\beta-1} p)$, for any β. Then $s = x$, the lower bound on the number of steps necessary to execute a jaw, is maximized when $s = x = \Theta(\log^{\beta} p)$ and therefore $y = \Theta(p \log^{\beta-1} p)$.*

Proof. As in Lemma 19, the objective is to maximize x. By Lemma 18, there exists some constant c such that

$$cx \geq y \log p/p.$$

Moreover, x and y are constrained such that

$$xy = p \log^{2\beta-1} p.$$

Substituting for x we obtain

$$cp^2 \log^{2\beta-2} p \geq y^2,$$

and taking square roots, we obtain

$$y \leq \sqrt{c} p \log^{\beta-1} p,$$

giving the promised bounds.

$\square$

We now calculate the total work W, the critical path D, and the makespan T_p for shark-tooth graphs with different values of β. There are three important cases: (1) $\beta \leq 0$, (2) $0 \leq \beta \leq 1$, and (3) $\beta \geq 1$. As we will see, Case (2) is the important case.

Theorem 21. *Consider a maximal shark-tooth DAG when $xy = \Theta(p \log^{2\beta-1} p)$.*

- *If $\beta \leq 0$, the makespan is $T_p = \Theta(D/\pi)$.*
- *If $0 \leq \beta \leq 1$, then the makespan is $T_p = \Omega(D \log^{\beta} p/\pi) = \Omega(W(\log^{1-\beta} p)/p\pi)$.*
- *If $\beta \geq 1$, then the makespan is $T_p = \Omega(W/p\pi)$.*

Proof. Case (1) follows from Lemma 18. In this case the makespan is just $\Theta(D/\pi)$, which matches the Graham lower bound of D/π. For Case (2), the total work W is the sum of the work on all the jaws plus the work on shark-tooth paths, which is

$$\begin{aligned} W &= \Theta(Dyx) \\ &= \Theta(Dp \log^{2\beta-1} p). \end{aligned}$$

The makespan is bounded below by the time to execute a jaw, $\Omega(x)$, times the number of jaws, $\Theta(D)$, which by Lemma 20 is

$$\begin{aligned} T_p &= \Omega(Dx/\pi) \\ &= \Omega(D \log^{\beta} p/\pi). \end{aligned}$$

Expressed in terms of the total work, W, the makespan is

$$T_p \geq \Omega(W(\log^{1-\beta} p)/p\pi). \quad (8)$$

Case (3) is just the Graham lower bound. Observe that (8) still holds, but is weaker than the Graham bound. $\square$

5. RELATED WORK

There are many papers in the literature that use eager and firing-squad scheduling. The principal idea of eager scheduling to run code so that parallel tasks are *idempotent*, that is, so that more than one processor can execute the same task at the same time. Thus, the computation is less likely to be delayed by a processor failing or running slowly. This technique was first proposed as an algorithmic method for transforming standard parallel (often PRAM) programs, which assume synchronization barriers, to run on hardware composed of asynchronous or fault-prone processors. See, e.g., [2, 3, 4, 5, 25, 26, 27, 28, 30] for examples of such transformations. Most of these algorithmic results focus on tightly coupled parallel programs, such as PRAM algorithms, or programs with synchronization barriers across all threads, unlike the current paper. Eager scheduling has subsequently been implemented in many parallel systems (see e.g., [7, 6, 8, 31]), ranging from networks of workstations to metacomputing and grid computing. More recent well-known distributed systems for exploiting idea computing cycles also benefit from eager scheduling and variations. Examples of such systems include SETI@home [1], the Globus Project [18], and grid-computing systems.

Two of the most notable research areas dealing with running parallel programs on processors of different speeds are *asynchronous parallel computing* and *scheduling on related processors*. Some asynchronous parallel computing papers include [3, 4, 17, 20, 24, 25, 26, 27, 28, 29, 30, 32]. In many of these works, the processor speeds are determined by an oblivious adversary, as in this paper. As we mentioned, most of the algorithmic work deals with tightly coupled parallel programs and parallel programs with synchronization barriers, as opposed to more general multi-threaded programs.

Executing a parallel program on processors of different speeds is also a common problem in scheduling theory. In this field there is an underlying assumption that the processors are *related*, i.e., they have different speeds but the speeds do not change. Because this problem is NP-hard [35] even for homogeneous processors, the scheduling community has concentrated on developing approximation algorithms for the makespan. Early papers introduce $O(\sqrt{p})$-approximation algorithms [22, 23], and more recent papers propose $O(\log p)$-approximation algorithms [15, 16, 13, 14]. These scheduling algorithms are unlikely to be directly applicable to running parallel programs because most of the algorithms are offline, only work for processors with unchanging speeds, require full-knowledge of the state of the system, and are too work-intensive. Moreover, the scheduling algorithms are not designed for the common case that $W/P \gg D$, and therefore optimize for uncommon cases.

Finally the quality of many of the scheduling algorithms are measured using the approximation ratio. Even in the *homogeneous setting*, i.e., when all processors run at the same speed, it is known that the approximation ratio may be misleading [11] by a factor as large as 2.

One exception to many of these rules is recent work on how to schedule Cilk multithreaded DAGs on different-speed processors [9, 10]. This result considers more general DAGs, optimizes for the common case that $W/p \gg D$, and considers processors that change speeds. However, the recent work does not consider highly variable processor speeds determined by an adversary, but rather more gently changing speeds where it makes sense for a processor to know its own speed.

6. CONCLUSION

Conventional wisdom in the parallel-computing community states that the extra dependencies caused by synchronization barriers lead to slower running times and should be avoided. We have proved that in firing-squad (eager) scheduling, this is not necessarily the case. Given an arbitrary DAG of precedence constraints, revealed online, we have proved that, somewhat surprisingly, adding a potentially dense set of dependencies can be provably advantageous. Specifically, in the worst case, with high probability, forcing synchronization barriers across all levels of the DAG gives an asymptotically shorter makespan compared to the schedule with no artificially-added dependencies. In particular, there is an advantage when the DAG has sufficient parallelism relative to the number of processors, but not overwhelming parallelism.

7. REFERENCES

[1] D. P. Anderson, J. Cobb, E. Korpela, M. Lebofsky, and D. Werthimer. Seti@home: an experiment in public-resource computing. *Commun. ACM*, 45(11):56–61, 2002.

[2] Y. Aumann, M. A. Bender, and L. Zhang. Efficient execution of nondeterministic parallel programs on asynchronous systems. In *Proceedings of the 8th Annual ACM Symposium on Parallel Algorithms and Architectures (SPAA)*, pages 270–276, 1996.

[3] Y. Aumann, M. A. Bender, and L. Zhang. Efficient execution of nondeterministic parallel programs on asynchronous systems. *Information and Computation*, 139(1):1–16, 25 Nov. 1997.

[4] Y. Aumann, Z. M. Kedem, K. V. Palem, and M. O. Rabin. Highly efficient asynchronous execution of large-grained parallel programs. In *Proceedings of the 34th Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 271–280, 1993.

[5] Y. Aumann and M. O. Rabin. Clock construction in fully asynchronous parallel systems and pram simulation. *Theoretical Computer Science*, 128:3–30, 1994.

[6] A. Baratloo, P. Dasgupta, V. Karamcheti, and Z. M. Kedem. Metacomputing with milan. In *Proceedings of the Eighth Heterogeneous Computing Workshop (HCW)*, page 169, 1999.

[7] A. Baratloo, P. Dasgupta, and Z. M. Kedem. Calypso: A novel software system for fault-tolerant parallel processing on distributed platforms. In *Proceedings of the 4th International Symposium on High Performance Distributed Computing (HPDC)*, pages 122–129, 1995.

[8] A. Baratloo, M. Karaul, Z. M. Kedem, and P. Wijckoff. Charlotte: metacomputing on the web. *9th International Conference on Parallel and Distributed Computing Systems (PDCS)*, 1996.

[9] M. A. Bender and M. O. Rabin. Scheduling Cilk multithreaded computations on processors of different speeds. In *Proceedings of the 12th Annual ACM Symposium on Parallel Algorithms and Architectures (SPAA)*, pages 13–21, July 2000.

[10] M. A. Bender and M. O. Rabin. Online scheduling of parallel programs on heterogeneous systems with applications to Cilk. *Theory of Computing Systems Special Issue on SPAA00*, 35:289–304, 2002.

[11] R. D. Blumofe and C. E. Leiserson. Scheduling multithreaded computations by work stealing. In *Proceedings of the 35th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 356–368, Santa Fe, New Mexico, Nov. 1994.

[12] R. P. Brent. The parallel evaluation of general arithmetic expressions. *Journal of the ACM*, 21(2):201–206, April 1974.

[13] C. Chekuri and M. A. Bender. An efficient approximation algorithm for minimizing makespan on uniformly related machines. In *Proceedings of the 6th Conference on Integer Programming and Combinatorial Optimization (IPCO)*, volume 1412, pages 383–393, 1998.

[14] C. Chekuri and M. A. Bender. An efficient approximation algorithm for minimizing makespan on uniformly related machines. *Journal of Algorithms*, 41:212–224, 2001.

[15] F. A. Chudak and D. B. Shmoys. Approximation algorithms for precedence-constrained scheduling problems on parallel machines that run at different speeds (extended abstract). In *Proceedings of the Eighth Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 581–590, New Orleans, Louisiana, 5–7 Jan. 1997.

[16] F. A. Chudak and D. B. Shmoys. Approximation algorithms for precedence-constrained scheduling problems on parallel machines that run at different speeds. *Journal of Algorithms*, 30(2):323–343, February 1999.

[17] R. Cole and O. Zajicek. The expected advantage of asynchrony. In *Proc. of the ACM Symposium on Parallel Architectures and Algorithms*, pages 85–94, 1989.

[18] I. Foster and C. Kesselman. Globus: A metacomputing infrastructure toolkit. *The International Journal of Supercomputer Applications and High Performance Computing*, 11(2):115–128, 1997.

[19] M. Frigo, C. E. Leiserson, and K. H. Randall. The implementation of the cilk-5 multithreaded language. In *Proceedings of the ACM SIGPLAN 1998 Conference on Programming Language Design and Implementation (PLDI)*, pages 212–223, 1998.

[20] P. B. Gibbons. A more practical PRAM model. In *Proc. of the 1st ACM Symposium on Parallel Architectures and Algorithms*, pages 158–168, June 1989.

[21] R. L. Graham. Bounds on multiprocessing timing anomalies. *SIAM Journal on Applied Mathematics*, 17(2):416–429, Mar. 1969.

[22] J. M. Jaffe. An analysis of preemptive multiprocessor job scheduling. *Mathematics of Operations Research*, 5(3):415–421, Aug. 1980.

[23] J. M. Jaffe. Efficient scheduling of tasks without full use of processor resources. *Theoretical Computer Science*, 12:1–17, Aug. 1980.

[24] P. C. Kanellakis and A. A. Shvartsman. Efficient parallel algorithms can be made robust. *Distributed Computing*, 5(4):201–217, 1992.

[25] Z. M. Kedem, K. V. Palem, M. O. Rabin, and A. Raghunathan. Efficient program transformation for resilient parallel computation via randomization. In *Proceedings of the 24th Annual ACM Symposium on the Theory of Computing (STOC)*, pages 306–317, May 1992.

[26] Z. M. Kedem, K. V. Palem, A. Raghunathan, and P. G. Spirakis. Combining tentative and definite executions for very fast dependable parallel computing. In *Proceedings of the 23rd Annual ACM Symposium on Theory of Computing (STOC)*, pages 381–390, May 1991.

[27] Z. M. Kedem, K. V. Palem, and P. G. Spirakis. Efficient robust parallel computations. In *Proceedings of the 22rd Annual ACM Symposium on Theory of Computing (STOC)*, pages 138–148, May 1990.

[28] S. C. Kontogiannis, G. E. Pantziou, P. G. Spirakis, and M. Yung. Robust parallel computations through randomization. *Theory of Computing Systems*, 33(5/6):427–464, 2000.

[29] G. Malewicz. Parallel scheduling of complex dags under uncertainty. In *Proceedings of the 17th Ann. ACM Symp. Parallelism in Algorithms and Architectures (SPAA)*, pages 66–75, 2005.

[30] C. Martel, A. Park, and R. Subramonian. Asynchronous PRAMs are (almost) as good as synchronous PRAMs. In *Proceedings of the 31st Annual Symposium on the Foundations of Computer Science (FOCS)*, pages 590–599, 1990.

[31] M. O. Neary and P. Cappello. Advanced eager scheduling for java-based adaptive parallel computing: Research articles. *Concurrency and Computation: Practice and Experience*, 17(7-8):797–819, 2005.

[32] N. Nishimura. Asynchronous shared memory parallel computation. In *Proc. of the 2nd ACM Symposium on Parallel Architectures and Algorithms*, pages 76–84, 1990.

[33] A. Panconesi and A. Srinivasan. Randomized distributed edge coloring via an extension of the Chernoff-Hoeffding bounds. *SIAM J. Comput.*, 26(2):350–368, 1997.

[34] A. Srinivasan. Distributions on level-sets with applications to approximation algorithms. In *Proceedings of the 42 Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 588–597, 2001.

[35] J. Ullman. NP-complete scheduling problems. *Journal Computing System Science*, 10:384–393, 1975.